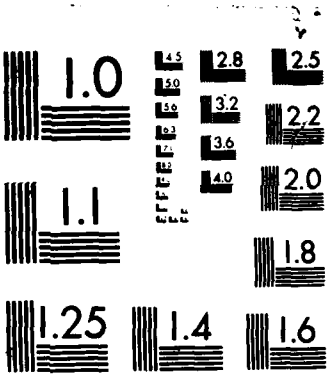
AD-A182 464 REDS: RESISTANCE EXTRACTION FOR DIGITALSSIMULATION(U) 1/1
STANFORD UNIV CA CENTER FOR INTEGRATED SYSTEMS
D STARK ET AL 1987 MDA903-83-C-0335
UNCLASSIFIED F/G 9/1 NL



REDS: RESISTANCE EXTRACTION FOR DIGITAL SIMULATION

Don Stark and Mark Horowitz

Center for Integrated Systems, Stanford University
Stanford, CA 94305

Abstract

This paper describes an extractor designed to produce resistance values for use in digital circuit simulation. REDS avoids resistance extraction on most nets in a design using a simple filter based on the perimeter and area values calculated by the capacitance extractor, allowing it to concentrate on areas where resistance may substantially affect circuit timing. Nets are extracted using a fast square counting algorithm, and simplified before output to remove spurious elements. REDS is designed to work on the Magic layout database.

1 Introduction

Parasitic resistances can substantially affect circuit performance, but are difficult to calculate efficiently. Most approaches fall into three categories: numeric calculation of Laplace's equation [6][1], 'lumped approximation'[9][3], and polygonal reduction [5][2][7]. Numeric solutions, though accurate, are slow; they are well suited only for small circuits. Lumped resistance values, where each node in a circuit has a resistance associated with it, are easily calculated from the node's perimeter and area, but are very inaccurate. Polygonal decomposition provides a good tradeoff between speed and accuracy, but is still expensive to run on large designs. Mori in [7] reduces this computation time by extracting hierarchically, but requires that explicit ports be defined for each subcell, which restricts allowed overlaps and makes additional work for the designer.

For logic simulation, the designer is primarily interested in knowing which resistors will substantially contribute to the RC delay of a given signal. This occurs when the interconnect resistance between two transistors is on the same order as the resistance of the transistor itself. Resistors should only be included for nets where this is the case; including additional components only slows down the simulator without substantially influencing the results.

The key to fast extraction for this timing is to extract only those nets which meet the above criterion. Unfortunately, the identity of these nets is not known until they are extracted. REDS avoids this problem by using the lumped resistance values to determine whether further extraction is required. The lumped values are essentially equal to the resistance of all geometries connected to a net stretched end to end. They form approximate upper bound (neglecting current crowding) on the maximum resistance between two points on a net. By comparing this value to the resistance of the largest transistor on a net, REDS avoids extracting most of the nets in a circuit.

REDS extracts the nets that fail the lumped resistance test by decomposing them into rectangles with unidirectional current flow. The resistance of each rectangle is calculated from its length and width. The maximum point-to-point resistance in the net is also determined; if this value is less than the transistor-resistance tolerance, the net is left untouched. If the resistance exceeds the tolerance, REDS estimates the time constant for the network and compares it to that of the transistor alone. If the time constant also exceeds the tolerance, REDS adds a resistor network to the extracted description.

REDS is part of the Magic layout system [9] and uses its corner stitched database database to efficiently estimate interconnect resistance. Before REDS is run, Magic's hierarchical extractor and netlist flattener are used to produce the lumped resistance values and a flat simulation file. The following section describes how REDS uses this simulation file along with the layout to select, extract, and simplify nets, while Section 3 provides some results from a large microprocessor design.

2 Implementation

2.1 Net selection

A resistance extractor's first task is to identify which nodes could have significant resistance. REDS finds the largest transistor with a source or drain connected to each node and compares the resistance of this 'driving' transistor with the lumped resistance of the node. The resistance of this driving transistor is calculated by dividing the length by the width and multiplying by the sheet resistance for the particular type of transistor. The lumped resistance is computed from the perimeter and area values calculated for each material by Magic during capacitance extraction. The formula for lumped resistance assumes the region is a rectangle and simultaneously solves equations for the rectangle's perimeter and area:

$$L \cdot W = area$$
$$2L + 2W = perimeter$$
$$L = (perimeter + \sqrt{perimeter^2 - area \cdot 16})/4$$
$$W = (perimeter - \sqrt{perimeter^2 - area \cdot 16})/4$$
$$resistance = (R/square) \cdot L/W$$

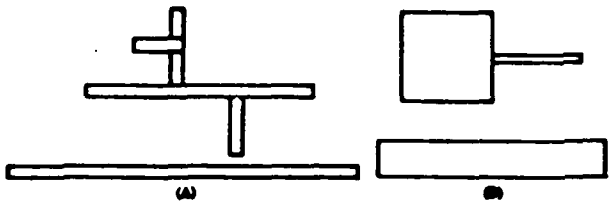


Figure 1: Lumped Resistance Approximation

The longer dimension is taken to be L because this gives the more conservative value. Two regions and their equivalent rectangles are shown in Figure 1. This process is approximately equal to stretching out the region into a single rectangle and gives an upper bound on the resistance between any two points in the network (Figure 1a) for networks of common width. When the width of the interconnect varies greatly, this approximation will not necessarily be an upper bound. Large, square regions of a highly resistive layer connected to long, thin areas (Figure 1b) will produce low lumped values. These regions are fortunately very rare in signal lines. The only notable exception is in pads, which contain large squares of metal and whose lumped resistance for the metal layer is correspondingly low. Even in pads, however, the lumped value is greater than the maximum point to point value because the resistance is dominated by polysilicon, not by metal, due to the disparity in sheet resistances.

If the resistance of the transistor is greater than this lumped resistance, no additional extraction on this particular net is necessary. Nets that fail this test are passed on to the next section of REDS, the net extractor.

2.2 Net Extraction

For each selected net, REDS reads and flattens the hierarchical layout description, storing the result in Magic's corner stitched database. Because Magic stores data in a canonical form (maximum horizontal rectangles) contacts tend to fracture contiguous regions into many subtiles. REDS reduces the number of tiles and simplifies extraction by eliminating contacts during a prepass over the layout. It records the location of contacts, then merges them into their constituent layers (Figure 2). The position of transistors is also noted, and a transistor data structure is allocated for each set of connecting transistor tiles. After this preprocessing, REDS is ready to begin actual extraction of the net.

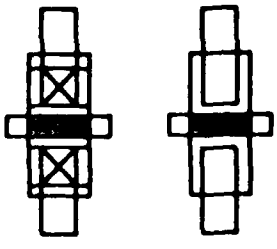


Figure 2: Contact Preprocessing

REDS calculates resistance by counting squares. For a given rectangular region the resistance is (R/square)*L/W, where L is taken in the direction of current flow. This method does well on long, narrow regions, but rather poorly on irregularly shaped ones. In the current application, however, the resistors of interest are formed primarily from long, thin regions. For a resistor to be on the same order of magnitude as the transistors to which it is connected, it must be many squares long because the sheet resistance of the interconnect is much less than that of the transistor. Irregularly shaped regions by definition can be only on the order of a few squares and need not be calculated with high precision.

The key problem in a square-counting resistance extractor is determining the direction of current flow. REDS simplifies this problem by assuming the current flow is either horizontal or vertical, a reasonable assumption because Magic only supports Manhattan geometries. The combination of orthogonal current flow and Manhattan geometries makes calculating W/L easy because every resistive region is a rectangle.

Current flow for an individual tile in Magic's database cannot be calculated without looking at the particular tile's neighbors. This can be seen in Figure 3. Magic's data structures prevent REDS from being able to determine current flow on a tile by tile basis. The use of maximum horizontal rectangles occasionally causes tall, narrow rectangles to be broken into long, thin strips. Individual tiles may have current flow perpendicular to their longest side. Fortunately, visiting a tile's neighbors in Magic's corner stitched database is fast and easy to do. REDS uses the pointers to adjoining tiles to search for connecting material. If it finds an adjacent tile of the same type, REDS checks to see whether the combined area is taller than it is wide. If it is, REDS assumes that current flows north-south. If it is not, then REDS continues to search. This process continues until either the combined region is taller than it is wide or no additional connecting tiles are found. If the final region is wider than it is tall, then the current flow is east-west. If not, current flow is north-south.

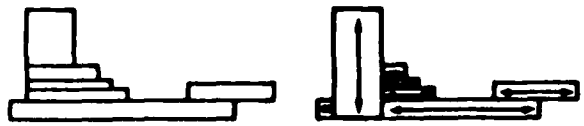


Figure 3: Determination of Current Direction

Once current flow for a given tile is known (note that current flow need not be same for all parts of the tile), REDS constructs resistors between the different sources/sinks of current in the tile. Possible connections include transistors, junctions with other tiles or with other regions of the same tile, and contacts. Resistors within a particular part of a tile are made by dividing the distance in the direction of current flow by the width or height of the region and multiplying by the sheet resistance. Each connection point becomes a node with one or more connecting resistors and/or transistors.

Resistor networks constructed in this fashion have several undesirable features which REDS tries to eliminate dynamically, including dangling connections, series resistors, parallel resistors, and 'triangles' of resistors. In each case, REDS eliminates the spurious nodes and resistors, then checks to see if the new network can be reduced further. This continues un-

til no further changes are possible. This simplification is done as soon as all geometries connected to a node are processed, keeping the size of the network as small as possible.

While it is reducing the network, REDS also keeps track of the resistance from each node to the starting transistor. After all nodes have been processed, the highest resistance node is found. If its resistance is less than that of the largest transistor, no resistive network is needed for this net, and the net is never output.

Comparing the maximum point to point resistance to the lumped value eliminates many resistors, but fails to catch one fairly common case. On nets with large fanout, the driver size must be large to handle the load; often, its resistance will be comparable to that of the interconnect leading to each gate. An example with fanout N=4 is shown in Figure 4a. The interconnect does not contribute significantly to the delay in this case, because only a small portion of the total load is driven through each interconnect resistor. For the special symmetrical case shown, the delay in Figures 4a and 4b are identical, with the delay due to interconnect being $R_t C_L/N$ instead of R_t/C_L. For large values of N, the interconnect delay is much less than a simple comparison of R_{trans} and R_t would indicate. More generally, timing depends not just on the relative resistor sizes, but also on the distribution of capacitance in the tree.

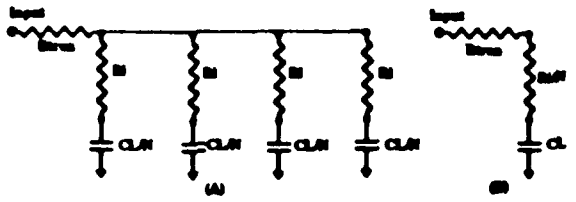


Figure 4: Effect of Fanout on Time Constant

REDS contains a final filter to recognize and remove large fanout nets based on their RC time constants. The filter is based on the estimates of delay in RC trees developed in [8]. The RC delay for a given node i can be approximated as the first order moment of the network's impulse response, $T_{Di} = \sum_k R_{ki} C_k$, where R_{ki} is the resistance from the input that is common between nodes k and i. Figure 5 shows the value of T_{Di} for node N3.

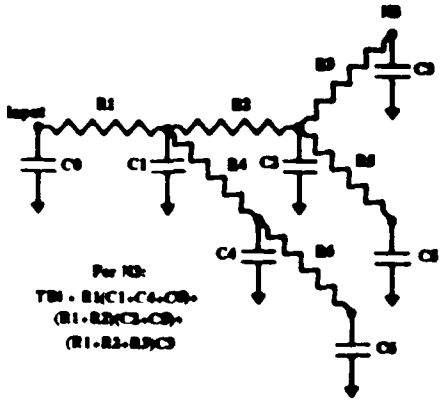


Figure 5: Calculation of TDi

Calculation of T_{Di} is fairly straightforward. First, the network's capacitance is distributed among its nodes. Next, REDS removes loops from the network by deleting the resistor in the loop that has the greatest resistance between its ends and the driving node. The resulting tree is then passed over twice. During the first pass, the capacitance for each node and its children is summed together. During the second, T_{Di} is calculated for each node by adding the product of node capacitance and the resistance between the node and its parent to T_{Di} for the parent. The largest value of T_{Di} is compared with the product of the transistor resistance and the total capacitance; if it exceeds the tolerance, then a RC network must be added to the circuit description.

2.3 Net Simplification

When a net's delay is greater than the tolerance, it must be written out. In its raw form, however, the net contains a plethora of nodes and small resistors. The detailed net contains more information than simulators and timing verifiers really need, and will slow them down substantially. This net must be simplified into something that retains the basic information but has fewer resistors and nodes.

A typical net looks something like Figure 6a, with many branches. REDS wants to modify the net to eliminate all resistors below the resistive tolerance. To do this, it starts at the node containing the driving transistor and walks down the tree, labeling resistors as visited and assigning them a direction. It knows it has reached the end when it gets to a node where all outward pointing resistors are terminals, i.e. connected to transistors only. If all the resistors connected to this terminal node are less than the tolerance, REDS deletes all but the largest resistor (Figure 6b). Once this branch is eliminated, the remaining tree has two resistor in series that can be combined. REDS can then walk back up the tree to the next branch. This continues until all the remaining resistors are greater than the tolerance (Figure 6c).

Networks reduced by the above method may still contain resistors less than the tolerance. To remove these remaining small resistors, REDS first sorts all resistors by size. The smallest resistor is combined with its smallest neighbor. This is continued until no resistors less than the tolerance remain.

The completed network is appended to the extract file of the design's root cell. Magic's extract format includes a 'killnode' command that eliminates a node and everything connected to

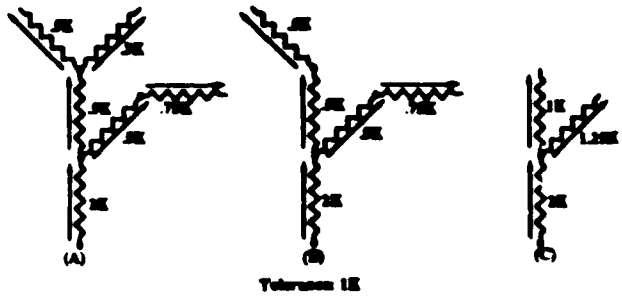


Figure 6: Net Simplification

it from the circuit's netlist; this is used to eliminate the old net and replace it with the new resistor-augmented one. REDS then repeats the process on the next net in the design.

3 Results

MIPS-X, Stanford's second generation RISC microprocessor [4], was used to test REDS. Excluding the instruction cache, MIPS-X contains 45,000 transistors and 19,000 nets. The instruction cache was not extracted because it is never included in switch level simulations. MIPS-X is designed in a 2μm, 2 level metal CMOS process. Technology parameters for MIPS-X are shown in Table 1.

Layer	Sheet Resistance (Ω/square)	Minimum Width (μm)
metal1	0.04	3
metal2	0.04	4
polysilicon	30	2
ndiffusion	40	3
pdiffusion	110	3
nfet	18000	3
pfet	36000	3

Table 1: Technology Parameters

Node failure rates are listed in Table 2 for a tolerance of 2 (i.e. the interconnect delay is at least 50% as large as the transistor delay.). All nodes except for the power supplies were included. The lumped resistance filter was effective in reducing the number of nodes to be extracted; only 20.0% failed. After extraction, only 0.4% of the nets needed to have resistors added, giving a 9% increase in circuit nodes. This is consistent with REDS' aim of adding the minimum number of extra components and nodes to correctly represent the net's charac-

Total Nets:	18880	
Lumped Failures:	3780	(20.1%)
Resistor Size Failures:	623	(3.2%)
Time Constant Failures:	83	(0.4%)

Type	Nets	Nodes	Resistors
Global Signals	5	1328	1354
Pads	58	264	206
Bus	11	33	22
Array Inputs	9	45	40
Totals	83	1670	1622

Table 2: Network Statistics

The nets that fail all checks fall into the 4 categories shown in Table 2. The largest number of resistors are created for global signals such as the clocks and a few frequently used control lines. The second major group of nets are located in the pads. Output pads fail because the final drivers have long polysilicon gates with substantial resistance. The next group contains part of one bus that was wired in polysilicon to save space. Finally, the inputs to structures such as PLA's and Weinberger arrays are run in polysilicon and also fail.

Running MIPS-X through REDS on a VAX-11/780 required about 137 minutes of CPU time. Interestingly, flattening the nets takes most of the time, while the actual extraction is relatively cheap. This suggests that speeding REDS up substantially will be fairly difficult because the database routines are already heavily optimized.

4 Conclusion

Digital simulation places two main constraints on a resistance extractor: extraction must be fast so that the turnaround time on changes is short, and the extracted network should only contain resistors that significantly affect performance. REDS runs quickly by reducing the number of nodes requiring extraction via a simple filter and extracting resistances using a square-based extraction algorithm that effectively utilizes Magic's corner-stitched database. REDS reduces the size of its output by distilling the original net into a smaller analogue that contains only the essential information. This combination makes REDS a useful tool in the simulation process.

5 Acknowledgements

Walter Scott (Lawrence Livermore National Laboratories), author of the Magic extractor, provided valuable advice and suggestions about integrating REDS into the Magic system. This work was partially funded by DARPA under contract number MDA 903-83-c-0335.

References

[1] Erich Barke, "Resistance Calculation From Mask Artwork Data by Finite Element Method", *Proceedings of the 22nd Design Automation Conference*, 1985, 306-311.

[2] J.D. Bastian, M. Ellement, P.J. Fowler, C.E. Hoang, and L. P. McNamee, "Symbolic Parasitic Extractor for Circuit Simulation (SPECS)", *Proceedings of the 20th Design Automation Conference*, 1983, 346-352.

[3] D. T. Fitzpatrick, "MEXTRA: A Manhattan Circuit Extractor", Electronic Research Lab Memo M82/62, Electronics Research Laboratory, University of California, Berkeley, January, 1982.

[4] Mark Horowitz, et al., "A 32b Microprocessor with On-Chip 2K Byte Instruction Cache", *1987 International Solid State Circuits Conference Digest of Technical Papers*, 30-31,326.

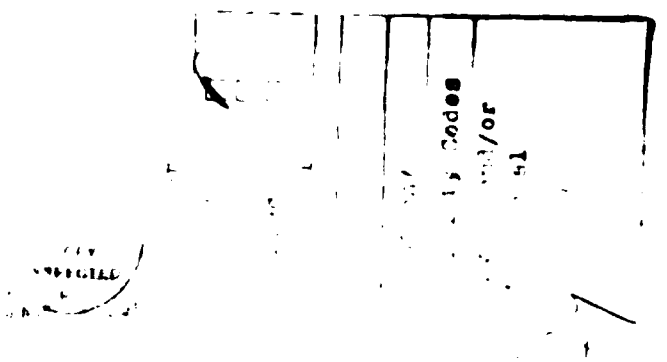
[5] Mark Horowitz and Robert W. Dutton, "Resistance Extraction from Mask Layout Data", *IEEE Transactions on Computer Aided Design*, Vol. Cad-2, No. 3, July 1983, 145-150.

[6] Steven P. McCormick, "EXCL: A Circuit Extractor for IC Design", *Proceedings of the 21st Design Automation Conference*, 1984, 616-623.

[7] Shojiro Mori and James A. Wilmore, "Resistance Extraction in a Hierarchical IC Artwork Verification System", *Digest of Technical Papers, IEEE International Conference on Computer Aided Design, 1985*, 196-198.

[8] Jorge Rubenstein, Paul Penfield, Jr. and Mark A. Horowitz, "Signal Delay in RC Tree Networks", *IEEE Transactions on Computer Aided Design*, Vol. Cad-2, No. 3, July 1983, 202-210

[9] Walter S. Scott and John K. Ousterhout, "Magic's Circuit Extractor", *Proceedings of the 22nd Design Automation Conference*, 1985, 286-292.



END

9-87

DTIC